IN THE SPECIFICATION

With reference to the published international application, i.e., WO 2004/046881, please amend/replace the below identified paragraph(s) as indicated, strikeout or double bracketed portions deleted, underlined items added:

➢Page 5, the first partial paragraph at line 10, please insert --of-- after "RenderMan in terms" so as to now read as follows:

RenderMan® is the name of a software program created and owned by Pixar that allows computers to render pseudo life-like digital images. RenderMan, a point-sampling global illumination rendering system and subject of U.S. Pat. No. 5,239,624, is the only software package to ever receive an Oscar® award from the Academy of Motion Picture Arts and Sciences. RenderMan clearly represents the current state of the art in pseudo-realistic point sampling software. On the other end of the spectrum, game consoles such as Sony PlayStation® or Microsoft X-Box® clearly do not exhibit the quality of realism found in RenderMan, but these hardware-based local illumination gaming appliances have a tremendous advantage over RenderMan in terms <u>of</u> speed. The realistic frames of animation produced by RenderMan take hours, even days, to compute, whereas the arcade-style graphics of gaming appliances are rendered at a rate of several frames per second.

➢Page 11 and continuing on Page 12, the descriptions with respect to FIGS. 2 and 3 are to be switched, so as to now read as follows:

FIG. 2 ~~is a schematic diagram of the operation of computer graphic process using an existing point-sampling process;~~ is a block diagram of a typical computer graphic process of rendering a digital image;

FIG. 3 ~~is a block diagram of a typical computer graphic process of rendering a digital image;~~ is a schematic diagram of the operation of computer graphic process using an existing point-sampling process;

➢Page 12, in the description with respect to FIG. 11, please insert --i.e., geometric function,-- after "system," so as to now read as follows:

FIG. 11 is a representation as FIG. 10, wherein an exemplary system, i.e., geometric function, and corresponding display are shown;

➢Page 14, the first partial paragraph please insert --(FIG. 1)-- after reference numeral "32" at line 3 thereof; delete reference number "50" at lines 7, 22, and 24; delete reference numeral "41" at line 12; delete reference numeral "43" at line 13; delete reference numeral "45" and "46" at line 14; delete reference numeral "47" at line 15; delete reference numeral "48" at line 16; replace "render farm 49" at line 19 with --network--; and insert reference numeral --51-- after "sequence together" at line 22, so as to now read as follows:

This grid technique is the real world analogy to the computer graphic process that forms the basis of modern day digital graphics. FIG. 2 shows the overall process of how a computer graphics system 40 turns a three dimensional digital representation of a scene 42 into multiple two-dimensional digital images 44. Just as the artist uses the cells 24 and 32 (FIG. 1) to divide the representation of an entire scene into several smaller and more manageable components, the digital graphics system 40 divides an image 44 to be displayed into thousands of pixels [[50]] in order to digitally display two-dimensional representations of three dimensional scenes. A typical computer generated image used by the modern motion picture industry, for example, is formed of a rectangular array of pixels 1,920 wide and 1,080 high. In a conventional digital animation process, for example, a modeler [[41]] defines geometric models [[43]] for each of a series of objects in a scene. A graphic artist [[45]] adds light, color and texture features [[46]] to geometric models of each object and an animator [[47]] then defines a set of motions and dynamics [[48]] defining how the objects will interact with each other and with light sources in the scene. All of this information is then collected and related in the scene database 42. A ~~render farm 49~~ network comprised of multiple servers then utilizes the scene database to perform the calculations necessary to color in each of the pixels [[50]] in each frame 44 that are sequenced together 51 to create the illusion of motion and action of the scene. Unlike the rectangular cells 32 on the artist's paper 30, a pixel [[50]] may only be assigned a single color.

➢Page 14, the second partial paragraph at line 27, please replace reference numeral "60" with reference numeral --22--, so as to now read as follows:

With reference to FIG. 3, like the artist via the viewing position [[60]] <u>22</u>, the system 40 of FIG. 2 simulates the process of looking through a rectangular array of pixels into the scene from the artists viewpoint. Current methodology uses a ray 62 that starts at the viewing position 60 and shoots through a location within pixel 50. The intersection of the ray with the pixel is called a point sample. The color of each point sample of pixel 50 is computed by intersecting this ray 62 with objects 64 in the scene. If several points of intersection exist between a ray 62 and objects in the scene, the visible intersection point 66 is the intersection closest to the origin of the viewing position 60 of the ray 62. The color computed at the visible point of intersection 66 is assigned to the point sample. If a ray does not hit any objects in the scene, the point sample is simply assigned a default "background" color. The final color of the pixel is then determined by filtering a neighborhood of point samples.

➢Page 21, line 1, please delete the second comma appearing in that line, so as to now read as follows:

Returning back again to the notion of point-sampling, and with reference now to FIGS. 6(a)-(f), FIG. 6(a) represents a single pixel containing four scene objects, with FIGS. 6(b)-(f) generally showing a point-sampling algorithm at work in furtherance of assigning the pixel a single color. As should be readily apparent, and generally intuitive, the color of the pixel might be some kind of amalgamation (i.e., integrated value) of the colors of the scene objects. In FIG. 6(b), only a single point sample is used, and it does not intersect with any of the objects in the scene; so the value of the pixel is assigned the default background color. In FIG. 6(c), four point samples are used, but only one object in the scene is intersected; so the value of the pixel is assigned a color that is 75% background color and 25% the color of the intersected scene object. In FIGS. 6(d), 6(e) and 6(f), additional point samples are used to compute better approximations (i.e., more accurate representations) for the color of the pixel. Even with the increased number of point samples in FIG. 6(e), two of the scene objects are not intersected (i.e., spatial aliasing: missing objects), and only in FIG. 6(f) does a computed color value of the pixel actually contain color contributions from all four scene objects. In general, point sampling[[,]] cannot guarantee that all scene objects contained within a pixel will be intersected, regardless of how many samples are used.

➢Page 24, line 21, please replace "William Walster, <u>Global Optimization</u> (publ. pending)" with --Eldon Hansen and William Walster, <u>Global Optimization Using Interval Analysis</u>, Second Edition--, so as to now read as follows:

The present invention, in all its embodiments, abandons point arithmetic and point-sampling techniques altogether, and instead turns to an interval analysis approach. First invented and published in 1966 by Ramon Moore, interval arithmetic is a generalization of the familiar point arithmetic. After a brief period of enthusiastic response from the technical community, interval arithmetic and interval analysis (i.e., the application of interval arithmetic to problem domains) soon lost its status as a popular computing paradigm because of its tendency to produce pessimistic results. Modern advances in interval computing have resolved many of these problems, and interval researchers are continuing to make advancements, see for example ~~William Walster, Global Optimization (publ. pending)~~ Eldon Hansen and William Walster, Global Optimization Using Interval Analysis, Second Edition; L. Jaulin et al., Applied Interval Analysis; and, Miguel Sainz, Modal Intervals.

➢Page 28, line 16 please delete "the" after "character," so as to now read as follows:

An output of the interval consistency solvers is indicated as pixel data (i.e., the task of the interval consistency solvers is to quantitatively assign a quality or character to a pixel). The pixel data output is ultimately used in image synthesis or reconstruction, vis-a-vis forwarding the quantitatively assigned pixel quality or character [[the]] to a display in furtherance of defining (i.e., forming) a 2-D array of pixels. For the parameterized system input of FIG. 11, a 2-D array of pixels, associated with a defined set of intervals, is illustrated.

➢Page 29, line 2, insert after "system" --, that is to say, a geometric function--, so as to now read as follows:

The solver, more particularly the most preferred components thereof, namely SCREEN, PIXEL, COVERAGE, DEPTH, and IMPORTANCE, are shown in relation to the input (i.e., dim and system, that is to say, a geometric function), callbacks (i.e., shader), and output (i.e., pixel data and display). The interrelationships between the individual most preferred elements of constituents of the solver, and the general temporal hierarchy between and among each, as well as their relationships between the callbacks (i.e., the shader) and the output (i.e., the display) are schematically shown in FIG. 12. As will be subsequently discussed in the flow schematics for each of the solvers, and as is appreciated by a reference to the subject figure, hierarchical, iterative sieving progresses, in nested fashion, from the screen solver to the importance solver, with each solver exporting a constraint for which the subsequent solver is to act in consideration thereof. Values from successively embedded solvers are returned as shown, the pixel solver ultimately bundling qualities or character of color, opacity, depth, and coverage, for instance, and "issues" such bundled information package (i.e., a pixel reflecting that scene object subtending same) to the display as shown in furtherance of synthesizing the 2-D array corresponding to the image plane.

➢Page 30, line 7 please replace "Chopping" with --Referring now to FIG. 14, chopping--, so as to now read as follows:

~~Chopping~~ Referring now to FIG. 14, chopping of the x-y image plane begins with an initial step analogous to that illustrated in FIG. 19(b). The idea is to parse the x-y image plane to dimensional equate to a pixel. As shown, in the event that initial chopping yields a sub divided x-y area more extensive than a pixel, more chopping is conducted, namely a preferential chopping. More particularly, the nature of the x-y image plane subunit (i.e., a rectangle) is assessed and characterized as being either "landscape" or "portrait". In the event the subunit is landscape, the x dimension is further split: in the event that the subunit is portrait, then the y dimension is then split. For each iterative step in x or y (see FIGS. 19(b) et seq., the arguments t, u, and v, are contracted so as to eliminate values thereof outside the specific or "working" x-y interval (i.e., with each iteration in x and y, it is advantageous to eliminate the t, u, and v values that are not contributing, and thereby potentially contribute to aliasing).

➢Page 32 line 19 replace "its" with --it--, so as to now read as follows:.

The depth solver, as detailed in FIG. 17, is essentially doing the job of FIG. 17(a). More particularly, DEPTH initially ascertains where in the z dimension, ultimately from the image plane (see FIG.

4 camera space), does the object surface, heretofore defined in x, y, t, u, v aspects, first appear or reside (i.e., in which depth cell), and thereafter step into space, via iterative cells, until the x, y, t, u, v object surface is no longer present in a cell (i.e., cell **X** of FIG. 17(a)). In furtherance thereof, the depth variable, more accurately, depth function, is initialized for all depth space, namely set to the infinite interval (z depth). Thereafter, t, u, v, contraction begins in the depth field (z0). Subsequently, there is a trivial accept/reject query as to whether there is in fact a depth component of the x-y parameterization, with searching commencing thereafter (z search). For each depth cell, the importance solver (i.e., the t, u, v, chopper wherein a set inversion is executed in t, u, v so as to contract same) is called upon, and it is necessary to next assess if the shader was invoked. If the shader is invoked (i.e., a first visible root is identified), the output of the shader are accumulated into the importance sums and the depth parsing continues in furtherance of accounting for all z components of the x-y object surface, if not, steps, on a cell by cell basis are "walked off." Although the parsing or chopping of z space has been described as a serial or loop type progression, [[its]] <u>it</u> is certainly amenable to recursive splitting, as the case of the x-y space.